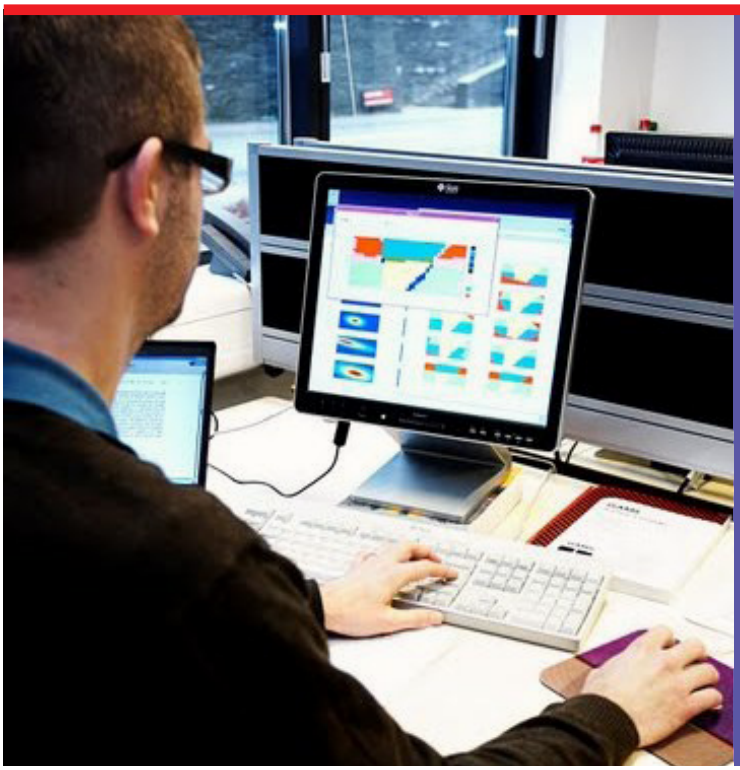
Greedy and metaheuristics for the offline scheduling problem in grid computing

Gamst, Mette

Publication date:
2010

Document Version
Publisher's PDF, also known as Version of record

Greedy and metaheuristics for the offline scheduling problem in grid computing



Report 2.2010

DTU Management Engineering

Mette Gamst
January 2010

DTU Management Engineering
Department of Management Engineering

Greedy and metaheuristics for the offline scheduling problem in grid computing

M. Gamst

DTU Management, Technical University of Denmark

Abstract

In grid computing a number of geographically distributed resources connected through the internet, are utilized as one computations unit. A number of grid servers control the activity on the grid, i.e., they decide, which job a resource should compute and when it should compute it. The grid servers run a scheduling algorithm to make such decisions. The offline scheduling problem consists of assigning jobs in a given time period to resources in advance, i.e., before that time periods begin. It is an $\mathcal{NP}$-hard problem. In this paper several heuristics for solving the offline scheduling problem are introduced.

The presented heuristics are five greedy algorithms, two local search algorithms and an adaptive large neighbourhood search. The heuristics are computationally evaluated. The results show that all heuristics find good solutions; the average gap between the heuristic and the optimal solutions is approximately 20%. The local search algorithms and the adaptive large neighbourhood search perform better than the greedy heuristics with respect to solution values. The greedy heuristics have much better running time than the local search algorithms and the adaptive large neighbourhood search. They are all capable of solving instances with up to 2000 jobs and 1000 resources, and the worst result values are within 30% of the optimal solutions. Thus, the heuristics for the offline scheduling problem in grid computing all give useful results both with respect to running times and to solution values.

1 Introduction

In grid computing a number of resources is used as one combined computations unit. The resources may be desktop computers, clusters, super computers, etc. Users log on to the grid in order to use the provided services. The idea behind grid computing is that it can be used to access applications, storage and computational power. Applications include all software - in this way the user only needs to install software on the home computer to log on to the grid. Using a grid providing storage means that the user can save all data on the grid rather than on a local computer. Finally, a grid providing computations power ensures that the user is capable of solving computational or data heavy jobs regardless of the performance power of the local computer. When grid computing emerged as a research field, the ambition was to implement a system as widespread as the power grid. Hence the name *grid* computing. The ambitions have yet to be met, though. Today grid computing systems consist of a limited number of resources and users and most systems are used by researchers to gain computational power.

Several implementations of grid computing exists, e.g., NORDUnet, Data Grid, The Worldwide LHC Computing Grid, and the Minimum intrusion Grid (MiG). Generally, a grid consists

of a number of users, resources and grid servers. The users request execution of certain jobs, the resources compute the jobs and the grid servers control all activity in the grid, i.e., which jobs to execute, the time for execution, the resources to use for execution etc. For more details on grid computing in general, including technical descriptions and an overview of existing grid projects, see the survey paper [6].

A grid typically consists of three sets of components; users, resources and grid servers. All components are connected through a Wide Area Network (WAN) and may thus be geographically distributed. The MiG is a good example of a grid and can be seen in Figure 1.

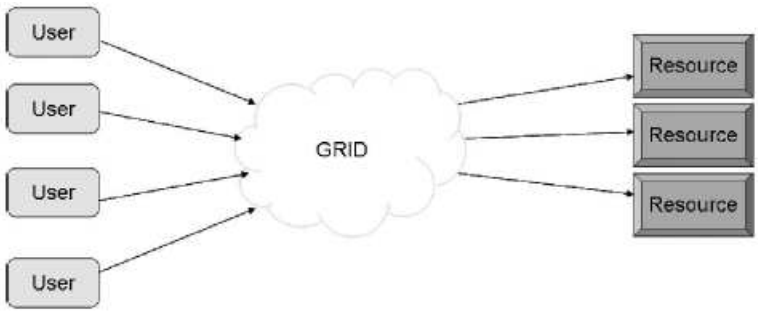


Figure 1: An abstract model of the Minimum intrusion Grid as presented by [3].

In the scheduling problem a number of jobs and resources are given. Each job consists of a number of data files distributed across the resources and each job is assigned a profit, which is paid when the job is executed. All resources are connected through the internet and bandwidths of all connection links are known. Each resource can execute at most one job at a time and job data files must arrive at the resource before execution can begin. The scheduling problem consists of assigning jobs to resources for execution subject to bandwidth and resource availability, such that the total profit of executed jobs is maximized.

Typically, job execution is handled online, i.e., the grid servers decide what to do with a job request the moment the request is received. See [18] for further work on the MiG and see [16] for online scheduling in MiG. Online scheduling, however, does not support services such as advance reservation, grid performance analysis, and avoidance of job starvation. An offline algorithm meets the desire for supporting these services. Given information on job requests, available resources, and expected bandwidth for a grid computing system in a given time space, the offline algorithm plans all activity in the grid in advance. In this way, it is possible to decide which resources to reserve for guaranteeing execution of a number of planned jobs. The algorithm can also be used to analyze the effect of adding/removing a number of jobs/resources etc. Yet another scope of the offline scheduling algorithm is to empty the job queue whenever the queue has reached a certain size. Emptying the job queue ensures that no job is queued forever, hence job starvation is avoided.

To the best of our knowledge not much literature exists on the scheduling problem with respect to bandwidth limitations in Grid Computing. A complexity proof and greedy heuristics for a simplified offline scheduling problem were presented by Marchal e.a. [13]. Agarwal e.a. [1] suggested an offline scheduler taking both job execution and data transmission into account. The method first schedules jobs to resources such that the total penalty of delayed job executions is minimized. Then the overall starting and end times of job schedules are

determined. Elghirani e.a. [7]proposed a tabu search algorithm, which schedules a queue of jobs. The neighbourhood of a solution consists of moving a scheduled job to a another available Grid resource. Often used moves are penalized to avoid move cycles. When no improvement has been reached in a certain time interval, the tabu list is cleared, a new random solution is found, and the tabu procedure starts all over. Varvaigos e.a. [17] considered job routing and scheduling to support Advance Reservations in the context of Grid Computing Advance reservations consist of scheduling data transmissions in a network and the task is to reserve the appropriate network resources. Varvaigos e.a. considered one data transmission request at a time, for which they found all optimal paths and then selected the "best" path according to a multi-cost objective and to available network resources.

We present five greedy heuristics, two local search algorithms and an adaptive large neighbourhood search for solving the offline scheduling problem in grid computing. The greedy heuristics use rather simple strategies for assigning jobs to resources. The main drawback of the greedy heuristics is that they do not consider the problem instance as a whole but concentrate on a job or a resource at a time. Hence, the two local search algorithms are introduced. The algorithms use one of the greedy heuristics to find a solution and to look in the neighbourhood of the best found solution. They thus try to take more than just a single job or resource into account. Their main drawback is that they are unable to escape local maximas, because they only accept solutions better than the current. This leads to the adaptive large neighbourhood search. It seeks to find good solutions by removing and adding jobs from the current solution via a greedy heuristic chosen in a roulette wheel manner, where each heuristic has a certain probability of being chosen. A solution may be accepted, despite not having the best solution value, in order to escape local maximas.

The heuristics are computationally evaluated. The results show that all heuristics find good solutions; the average gap between the heuristic and the optimal solutions is approximately 20%. The local search algorithms and the adaptive large neighbourhood search perform better than the greedy heuristics with respect to solution values. The greedy heuristics have much better running time than the local search algorithms and the adaptive large neighbourhood search. They are all capable of solving instances with up to 2000 jobs and 1000 resources, and the worst result values are within 30% of the optimal solutions. Thus, the heuristics for the offline scheduling problem in grid computing all give useful results both with respect to running times and to solution values.

This paper is structured as follows. First in Section 2, the offline scheduling problem is formally described and it is shown to be $\mathcal{NP}$-hard. Then the heuristics are introduced. The greedy heuristics are presented in Section 3, the local search algorithms in Section 4 and the adaptive large neighbourhood search algorithm in Section 5. The heuristics are computationally evaluated in Section 6, where test instances are also described and information on preprocessing and parameter tuning is presented. Finally, Section 7 contains final remarks on the work.

2 Problem Description

This section contains a formal description of the offline scheduling problem. First, some assumptions are made and the contents of a problem instance are formalized. The problem description follows and finally the complexity of the problem is presented.

Only one grid server is assumed as communication between servers in the grid is prede-

termined. The grid server is considered as a resource unable to execute jobs but capable of holding job data. Also, it is assumed that the only data to be sent between resources is job data. This abstraction is fair as bandwidth usage of job requests and job result files are insignificant and can thus be ignored.

A problem instance includes the following information:

- A list of jobs J to be executed. The number of jobs is denoted $|J|$. Each job, $j \in J$ includes the following information:
  - A time window, (a_j, b_j), i.e., the time at which job j is submitted (start time, a_j) and a time stamp at which it must be either executed or discarded (end time, b_j)
  - The profit, c_j, for job j.
  - List of input files, i.e., a list of the required job files, the position of the job files (where from data is to be copied) and the size of each job file. The amount of data located on resource r for job j is denoted p_j^r. The total size of the job is denoted S_j
  - Estimated execution time, Q_j
- A set of resources, R. The number of resources is denoted $|R|$. A time window (a_r, b_r) is attached to each resource, $r \in R$
- Bandwidth availabilities out of and into each resource and between each pair of resources in the grid. Bandwidths may vary with time. Available bandwidth at time t for incoming data at resource r is denoted $d_{r^-}^t$, available bandwidth at time t for outgoing data at resource r is denoted $d_{r^+}^+$, and available bandwidth at time t for data between resources r, k is denoted d_{rk}^t.

The goal is to maximize the profit of executed jobs. Before a job can be executed, all job files must be copied to the executing resource. Also, bandwidths must be obeyed when sending data. Each resource is capable of receiving data while executing another job, but can execute at most one job at a time.

For each instance, the set of all time slots is denoted T and the total time of the instance, i.e., the difference between the latest end time and the earliest start time for all jobs and resources, is denoted $|T|$.

2.1 Mathematical formulation

The offline scheduling problem is formulated mathematically on a graph G. The graph G consists of a set of nodes V, and a set of edges E. The set V is a union of the set of jobs and resources, i.e., $V = J \cup R$. Each edge $(i, k) \in E$ connects resource i with resource k.

To simplify notation, the time window $[a_{ik}, b_{ik}]$ is introduced, where $a_{ik} = \min\{a_i, a_k\}$ and $b_{ik} = \min\{b_i, b_k\}$, for some $i, k \in R \cup J$. For further notational convenience, two sets are introduced: J_t and R_t. The set J_t consists of jobs, j, with $a_j \leq t \leq b$. Similarly, the set R_t consists of resources, r, with $a_r \leq t \leq b$.

The mathematical model includes two types of variables; $x_j^{tr} \in \{0, 1\}$ and $f_{ir}^{tj} \in \mathbb{Q}^+$. If $x_j^{tr} = 1$ then job $j \in J$ is executed on resource $r \in R$, with execution beginning at time $t \in T$. If $x_j^{tr} = 0$ then the job is not executed on the resource with this beginning time. The

non-negative variable $f_{ir}^{tj} \in \mathbb{Q}^+$ denotes the amount of data on edge $(i,r) \in E$ for job $j \in J$ at time $t \in T$.

The offline scheduling problem can now be formulated mathematically. An edge-based presentation is:

$$\max \quad \sum_{r\in R}\sum_{j\in J}\sum_{t\in[a_{rj},b_{rj}-Q_j]} c_j x_j^{rt} \tag{1}$$

$$\text{s.t.} \quad \sum_{r\in R}\sum_{t\in[a_{rj},b_{rj}-Q_j]} x_j^{rt} \le 1 \qquad \forall j \in J \tag{2}$$

$$\sum_{j\in J_t}\sum_{i\in R_t\setminus\{r\}} f_{ri}^{tj} \le d_{r^+}^t \qquad \forall r \in R,\, \forall t \in [a_r, b_r] \tag{3}$$

$$\sum_{j\in J_t}\sum_{i\in R_t\setminus\{r\}} f_{ir}^{tj} \le d_{r^-}^t \qquad \forall r \in R,\, \forall t \in [a_r, b_r] \tag{4}$$

$$\sum_{j\in J_t} f_{ri}^{tj} \le d_{ri}^t \qquad \forall r, i \in R,\, \forall t \in [a_{ri}, b_{ri}] \tag{5}$$

$$x_j^{rt} = 1 \Rightarrow \sum_{t'=a_{ij}}^{\min\{b_i,t-1\}} f_{ir}^{t'j} = p_j^i \qquad \forall j \in J, \forall i, r \in R, \forall t \in [a_{rj}, b_{rj} - Q_j] \tag{6}$$

$$x_j^{rt} = 1 \Rightarrow \sum_{j'\in J\setminus\{j\}} \sum_{t'=t}^{\min\{t+Q_j,b_{jr}-Q_j\}} x_{j'}^{rt'} = 0 \quad \forall j \in J, \forall r \in R, \forall t \in [a_{rj}, b_{rj} - Q_j] \tag{7}$$

$$x_j^{rt} \in \{0,1\} \qquad \forall t \in [a_{rj}, b_{rj} - Q_j],\, \forall j \in J,\, \forall r \in R$$

$$f_{ir}^{tj} \ge 0 \qquad \begin{array}{l} \forall j \in J,\, \forall r, i \in R : p_j^i > 0, \\ \forall t \in [a_{rji}, \min(b_i, b_{rj} - Q_j - 1)] \end{array}$$

The objective function (1) maximizes the profit of the executed jobs. The first constraint (2) says that each job can be executed at most once. Constraints (3) and (4) make sure that in- and outgoing bandwidth limitations are obeyed and constraint (5) ensures that connection capacities are obeyed. All job data must be received before execution time (6). Constraint (7) says that a resource can execute at most one job at a time. The two bounds ensure that variables take on appropriate values.

2.2 Complexity

The problem is $\mathcal{NP}$-hard, which is shown by reduction from the $\mathcal{NP}$-hard *knapsack problem*. The data transmission part only adds to the complexity of the scheduling problem, thus it suffices to show that the job assignment problem is $\mathcal{NP}$-hard.

In the knapsack problem, a knapsack and a set of items are given and each item has an attached profit. The goal is to pack items into the knapsack such that the total profit of packed items is maximized. For a survey of the knapsack problem and corresponding solution methods, see [14].

Consider the knapsack problem instance where a knapsack is to be packed. This is equivalent to assigning jobs to a resource such that the total profit of executed jobs is maximized. Hence, a solution to the job assignment problem is applicable on the knapsack problem. Hence, the job assignment problem and the offline scheduling problem are $\mathcal{NP}$-hard.

A number of heuristics are adapted to the offline scheduling problem. The heuristics are divided into three classes: greedy heuristics, local search algorithms and an Adaptive Large Neighbourhood Search (ALNS) algorithm.

3 Greedy heuristics

Five greedy heuristics are presented. They all seek to assign jobs to resources through a few simple steps.

The heuristics need to take data transmission into account, i.e., the problem of sending job data to the executing resource before execution begins. In the following section the data transmission problem is described in detail and a heuristic for solving the problem is presented.

3.1 The data transmission problem

All resources are directly connected so when sending job data from one resource to another, the direct link is used. Sending job data is denoted the data transmission problem. In this Section, the data transmission problem is transformed into the Linear Multi-commodity Flow Problem (MFP), which is polynomially solvable.

For MFP, a number of commodities and a graph are given. The graph consists of a number of nodes and edges between the nodes. Edges are assigned a capacity and a cost. Each commodity consists of a start node, an end node and an amount of flow to be send between the start and end nodes. The MFP is to send all commodity data without violating edge capacities such that the total cost is minimized, see [2].

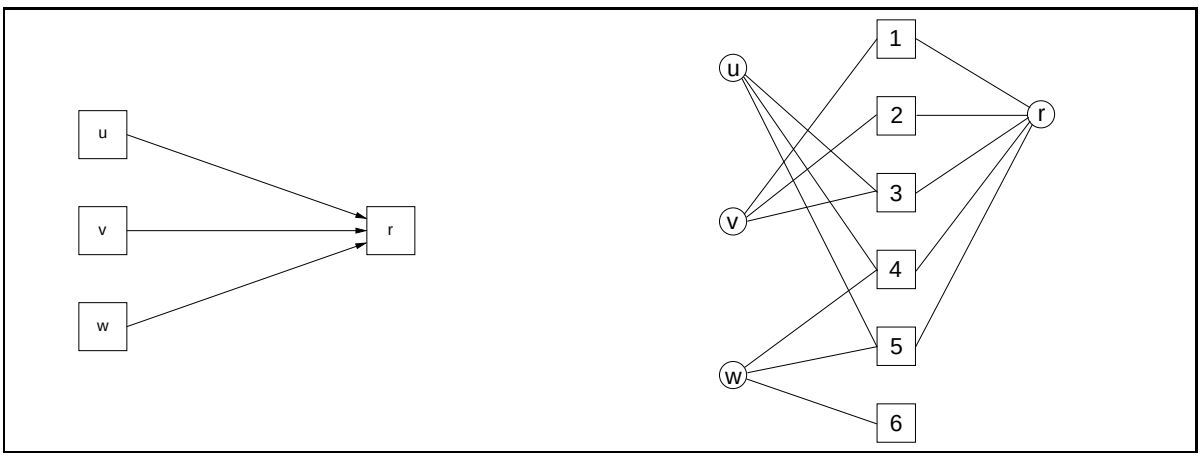
Looking at the data transmission problem for a job, j, and an executing resource, r, we need to send job data for j from several resources, $r_1, r_2, \ldots, r_k$ to r before execution can begin.

Commodities are introduced: the resources $r_1, r_2, \ldots, r_k$ are now start nodes and r is the end node. The flow to send between each pair of start and end nodes is the job data. Hence, each commodity has start node $\in \{r_1, r_2, \ldots, r_k\}$ and end node r and has flow equal to the corresponding job data. In this way, the data transmission problem corresponds to the MFP.

Data can be sent at different time stamps in the data transmission problem. When transforming this into the MFP, we get a MFP working over time. To handle the time aspect in MFP, the graph is transformed into a *time expanded* graph as done for the single-commodity flow problem over time by Ford and Fulkerson, see [8, 9].

In the time expanded graph, sources and target are connected through a set of added *time nodes*; each time node represent a time stamp. This is illustrated in Example 1. Bandwidth limitations are represented as edge capacities. Consider the instance from Example 1: the edge going from resource u to time node 1 has bandwidth $\min\{d^1_{u^+}, d^1_{ur}\}$, and the edge going from time node 1 to resource r has capacity $d^1_{r^-}$. These examples of edge capacities illustrate how to find capacities for all edges in the graph. Now, solving the data transmission problem on the time expanded graph corresponds to solving the MFP.

Even though the MFP is polynomially solvable, larger instances can be difficult to solve. In the literature large instances of the MFP are typically solved using Lagrangian methods, partition methods, decomposition techniques, dual ascent algorithms, bundle methods, interior point methods, etc., see [5] and [11] for surveys of the problem, and [12] for a review of solution techniques. Small instances are typically solved using the Simplex algorithm. No straightforward combinatorial solution algorithm exists. We choose to solve the data transmission



Example 1: Two graphs are seen. The graph on the left-hand side represents an instance consisting of three source resources: u with time window $[3, 5]$, v with time window $[1, 3]$ and w with time window $[4, 6]$. The target node, r, has time window $[1, 5]$. The figure on the right-hand side shows the time expanded graph. Nodes representing times $1, \ldots, 6$ are introduced, and u, v, and w are connected to r via nodes, representing time slots where both parts are available.

problem heuristically in order to keep the greedy heuristics simple. The greedy heuristics and the data transmission algorithm are described in detail in the following.

3.1.1 Heuristic for the data transmission problem

The data transmission problem is solved heuristically for each pair of job and resource (j, r). Starting from the top of the list of resources holding job data for j, the data is greedily sent to r. Data transmission from a resource starts as early as possible with respect to time windows and bandwidth availabilities, and the transmission continues until all data is sent, if possible.

The running time of the algorithm is $\mathcal{O}(|R||T|)$ because the algorithm in worst case investigates all time spaces for all resources.

3.2 First come, first serve

This heuristic assigns the first available job to the first available resource, on which job execution finishes first. The heuristic does not take profits for executed jobs into account.

The first come first serve algorithm has running time $\mathcal{O}(|J||R|^2|T|)$; in worst case, the data transmission algorithm is run for each pair of job and resource.

3.3 Best first

Starting with the job with highest profit, each job is assigned to the resource at which the job execution finishes first.

The best first heuristic has running time $\mathcal{O}(|J| \log |J| + |J||R|^2|T|)$; jobs are sorted according to profit, and in worst case the data transmission algorithm is run for each pair of job and resource.

3.4 First fit

This heuristic first computes the start execution time for all jobs on a resource. The job with lowest start execution time is then assigned to the resource. If two jobs have the same execution finish time, then the job with highest profit is executed. This is repeated until it is not possible to assign any jobs to any resources.

The first fit heuristic has running time $\mathcal{O}(|J|^2|R|^2|T|)$: in worst case, the data transmission algorithm is run for each resource and each pair of jobs.

3.5 Best fit

This heuristic investigates the time usage for all jobs on each resource. The job with smallest time usage for data transmission and execution is assigned to the resource. If two jobs have the same time usage, then the job with highest profit is executed.

The best fit heuristic has running time $\mathcal{O}(|J|^2|T|^2|R|^2)$; the data transmission time is run for each pair of jobs and for each resource at each possible starting time.

3.6 Random fit

Jobs are chosen randomly and if possible assigned to a resource. Profits are not taken into account. The random fit heuristic has running time $\mathcal{O}(|J||R|^2|T|)$; in worst case the data transmission algorithm is run for each pair of job and resource.

4 Local search algorithms

The greedy heuristics focus on a single job or a single resource at a time rather than taking all or some of the remaining into account. To compensate for this drawback, two local search heuristics are implemented. A local search algorithm attempts to improve the current solution by searching the neighbourhood. Let $N(sol)$ denote the neighbourhood of some feasible solution sol. Then a general local search algorithm is illustrated in Algorithm 1.

local_search(k)

1: $sol \leftarrow$ an arbitrary feasible solution in S
2: **while** ($\exists sol' \in N(sol)$ such that $profit(sol') > profit(sol)$) **do**
3: $\quad sol \leftarrow sol'$
4: **end while**
5: **return** sol

Algorithm 1: Illustration of a general local search heuristic

Given a feasible solution executing l jobs, the neighbourhood is defined by the set of other feasible solutions, executing $l - k$ of the same jobs. That is, in an attempt to improve the current solution, k executed jobs are temporarily left out - or *exchanged* - to make room for other jobs. This k-exchange approach is in the literature also known as the swapping method. The k-exchange, local search algorithm is used as an approximation algorithm for the facility location problem by [4]. For a historic overview of the method refer to this article.

Two local search algorithms are implemented:

- 1-exchange local search. Here $k = 1$. Exactly one executed job in the current solution is blocked, i.e. the job cannot be executed in the next solution.
- k-exchange local search. Up to k, where $1 \leq k \leq l$, executed jobs in the current solution are blocked, i.e. the jobs cannot be executed in the next solution.

Note, that the local search heuristics need a start solution with at least one executed job to work. The exchange is performed until either a fixed upper bound on the number of iterations (step 2-4 in Algorithm 1) is reached or the solution does not improve. The reason for setting an upper bound on the number of iterations is that otherwise the running time may be exponential. In worst case all combinations of jobs may be exchanged per resource and the number of exchanges is $\mathcal{O}(|R| \cdot \sum_{i=1}^{|J|} \binom{|J|}{i})$. Let ub be the upper bound on the number of iterations. Then the local search heuristics have polynomial running time $\mathcal{O}(ub \cdot heur)$, where $heur$ is the (polynomial) running time of the used greedy heuristic.

The local search algorithms need a method for finding a start solution and for adding non-executed jobs to the current solution. All of the greedy heuristics introduced earlier are usable and the best choice is found through computational evaluation.

5 Adaptive large neighbourhood search

The local search algorithms may get stuck in local maximas. This is due to the algorithms only accepting solutions better than the current and only looking in the neighbourhood of the current solution. The adaptive large neighbourhood search is capable of escaping local maximas and is thus implemented. The Adaptive Large Neighbourhood Search (ALNS) algorithm is presented for the Pickup and Delivery Problem with Time Windows by [15]

The ALNS is adapted to the offline scheduling problem in grid computing. Given a start solution, a subset of the executed jobs is removed from the solution using a removal heuristic. Some of the non-executed jobs are then inserted into the solution using an insertion algorithm. The solution is saved if it is accepted according to certain criteria. This procedure is repeated until some stop criterion is met. Both several removal heuristics and several insertion heuristics are implemented. Each heuristic is rewarded according to how it performs and the probability of a heuristic being chosen depends on the corresponding received reward. Performance is measured both according to solution values and to new solutions.

Algorithm 2 shows the framework for ALNS as an offline scheduling algorithm. The framework does not specify how removal and insertion heuristics are chosen. We choose to set the stop criterion, as seen in step 2 in Algorithm 2, to whether or not the best solution has been improved. If not, then the stop criterion is met. The number of times, the heuristic and insertion heuristics are run in an iteration (step 3-12 in Algorithm 2) is set through parameter tuning.

5.1 Removal heuristics

The removal heuristics use the current solution to select which jobs to remove. Furthermore, the removal heuristics take some integer, k, as input, which defines the number of jobs to remove. The input k is set through parameter tuning and is never larger than the number of executed jobs in the current solution.

```
ALNS(s ∈ {Solutions}, k)
 1: Solution s_best ← s
 2: while (stop-criterion is not met) do
 3:   for (A fixed number of iterations) do
 4:     s' ← s
 5:     remove k jobs from s'
 6:     reinsert a subset of the non-executed jobs into s'
 7:     if (profit(s') > profit(s_best)) then
 8:       s_best ← s'
 9:     end if
10:     if (accept(s', s)) then
11:       s ← s'
12:     end if
13:   end for
14:   Reward removal and insertion algorithms according to performance
15: end while
16: return s_best
```

Algorithm 2: The adaptive large neighbourhood search framework.

5.1.1 Profit based heuristic

The algorithm removes the k jobs with smallest profit from the current solution. The algorithm does not take time usage into account. Running time is $\mathcal{O}(|J| \log |J|)$, i.e., the time it takes to sorting the executed jobs. Sorting is necessary to decide which k jobs to remove.

5.1.2 Profit and time based heuristic

Let W_j be a variable which denotes the time usage for sending job data for job j. For each executed job, $j \in J$, the term $c' = c_j/(Q_j + W_j)$ is calculated, i.e., the profit per used time slot is found. Time usage includes time spent on sending job data and on executing the job. Jobs with k lowest profit per time slot are removed from the current solution. Again the running time is the sort time, $\mathcal{O}(|J| \log |J|)$.

5.1.3 Random removal

This algorithm selects k random jobs from the current solution, and then removes the jobs. Running time is $\mathcal{O}(k)$.

5.2 Insertion heuristics

The previously introduced greedy heuristics can be used for inserting jobs. The insertion heuristics, however, do not build a solution from base, but rather continues work on a partial solution.

The heuristics presented in Section 3 are slightly modified. Executed jobs are not reinserted and edge capacities and resource availability are modified to reflect data transmission and job execution of the already executed jobs. Furthermore, the heuristics attempt to assign a job,

which has just been removed from the solution by the removal heuristic to another resource than that previously used.

5.3 Adaptive weight adjustment

The ALNS is adaptive as the choice of removal and insertion heuristics is adjusted in each iteration of the algorithm. Each heuristic is assigned a probability for selection. Given h heuristics with weights $w_i,\ i \in \{1, 2, \ldots, k\}$, heuristic l is selected with probability:

$$\frac{w_l}{\sum_{i=1}^{h} w_i}$$

The weights w_l are set automatically according to how well heuristic l performs. For each iteration of the ALNS (step 2-12 in Algorithm 2) w_l is recalculated. Given heuristic l and iteration i, then w_l for iteration $i+1$ is calculated as:

$$w_{l,i+1} = w_{li}(1-\rho) + \rho\frac{S_l}{\alpha_l}$$

where ρ is the *reaction factor* controlling how quickly the weight adjustment algorithm reacts to changes in the solution values found by the heuristics. The reaction factor is set through parameter tuning. The constant α_l denotes the number of times heuristic l has been used in the last iteration and S_l is the score of heuristic l obtained during the last iteration. The score is calculated according to three situations:

- The last remove-insert operation resulted in a new global best solution.
- The last remove-insert operation resulted in a solution that has not been accepted before. The profit of the new solution is better than the profit of the current solution.
- The last remove-insert operation resulted in a solution that has not been accepted before. The profit of the new solution is worse than the profit of current solution, but the solution was accepted.

The sizes of the three different scores are found using parameter setting. Solutions are stored in a hash table to ease the check for whether a solution has already been found or not.

Simulated annealing is used to accept solutions. It can be beneficial to accept a solution, even if it has worse solution value than the current solution. The reason for this is that the algorithm may be stuck in a local maxima and accepting a worst solution can lead the algorithm from a local maxima towards a global maxima. Let s be the current solution, and let $\tau > 0$ be the temperature. Then solution s' is accepted with probability $e^{-\frac{f(s')-f(s)}{\tau}}$. The cooling is $\tau = \tau \cdot (1-c)$; where the *cooling rate* $0 \leq c \leq 1$ is found via parameter tuning. The start temperature, τ_{start}, is calculated from the initial solution. The profit of the initial solution is denoted z'. The start temperature is set such that a solution, which is w percent worse than the current, is accepted with probability 0.5. The *start temperature control parameter*, w, is set via parameter tuning.

6 Computational Results

The proposed heuristics are tested and compared with each other. First, problem instances have been generated. Details regarding the data generation are presented in the following. Next, preprocessing is performed to limit the solution space. The preprocessing steps are presented later in this section.

The local search heuristics and the ALNS require parameter tuning. The algorithms are tested on different parameter settings and the best settings are chosen. Finally, all heuristics are computationally evaluated and the results are presented and analyzed. All tests are run on a 2.66 GHz Intel Xeon machine with 8 GB RAM. Note that CPU times in the following refer to using one core.

6.1 Data generation

This section contains an overview of how test data is generated.

The instances are generates such that the number of jobs and resources vary. All instances are generated to reflect activity in a grid computing system within a 24 hour period. Resources are generated such that their start and end time lie within the 24 hour time span. Both the start and end time are randomly generated within this bound. Furthermore, the end time lies at least one time slot later than the start time. Bandwidth availabilities are set randomly between 0 and 10. Jobs are generated such that job data is distributed across the resources, and such that each job data source holds at most five units of job data. The job start time lies within the 24 hour time span and the end time is set to be twice the size of the job after the start time. To show any connection between problem instance size and the complexity of the scheduling problem, instances with 100, 200, 500, 1000 or 2000 jobs and 10, 20, 50, 100, 200, 500 or 1000 resources are generated.

6.2 Preprocessing

The goal of preprocessing is to limit the size of a problem instance. Here, the preprocessing consists of the following two steps:

Job data source availability
: If a resource containing job data is not available in the same time space as the job then the job cannot be executed. The running time of this preprocessing step is $\mathcal{O}(|J||R|)$ as it, in worst case, needs to look at all resources for each job.

Job data source bandwidth
: If a resource containing job data does not have enough available bandwidth to send out all data in time for latest possible execution time and if the resource cannot execute the job itself, then the job cannot be executed. The running time of this preprocessing step is $\mathcal{O}(|J||R||T|)$ as it, in worst case, needs to consider all time slots for all resources for each jobs.

The preprocessing steps remove the jobs, which it has established cannot be executed. Hence, the preprocessing potentially reduces the problem instance and it saves the heuristics from considering jobs, which never can be included in any feasible solution.

6.3 Parameter tuning

The parameters for the local search and the ALNS algorithms are tuned. To limit the scope of the parameter tuning process only a subset of the instances has been solved. The 1-exchange and ALNS algorithms are tested on instances with 200 and 500 jobs, while the k-exchange algorithm is tested on instances with 100 and 200 jobs.

6.3.1 The local search algorithms

It is necessary to decide the maximum number of iterations for both local search algorithms. If an upper bound is not set then the algorithms have potentially exponential running times. Furthermore, the k-exchange algorithm needs an upper bound on the number of jobs to exchange at a time. The parameters are set through the following tuning.

1-exchange algorithm

The maximal number of iterations in the 1-exchange algorithm is to be set. First come first serve is used as heuristic and the maximal number of iterations is set to 5, 10, 20, 50 and 100, respectively. Results are seen in Table 1.

		5 iter.		10 iter.		20 iter.		50 iter.		100 iter.	
Jobs	Res.	Result	Time	Result	Time	Result	Time	Result	Time	Result	Time
200	10	142	0.17	142	0.17	142	0.17	142	0.17	142	0.16
200	20	154	0.42	154	0.41	154	0.41	154	0.42	154	0.41
200	50	160	3.08	160	3.05	160	3.12	160	3.11	160	3.08
200	100	168	6.84	168	6.83	168	6.75	168	6.68	168	6.64
200	200	194	57.64	194	58.01	194	57.99	194	57.57	194	58.35
200	500	144	220.41	144	220.39	144	220.39	144	220.29	144	221.01
200	1000	162	1315.66	162	1274.50	162	1288.40	162	1271.15	162	1269.95
500	10	290	1.23	290	1.23	290	1.24	290	1.22	290	1.22
500	20	378	3.17	378	3.14	378	3.15	378	3.15	378	3.16
500	50	422	19.89	422	19.95	422	20.04	422	20.07	422	20.02
500	100	426	76.97	426	77.00	426	76.90	426	77.32	426	77.53
500	200	374	275.43	374	276.13	374	275.90	374	279.65	374	276.10
500	500	426	1492.35	426	1501.09	426	1495.79	426	1496.56	426	1493.15
500	1000	398	7558.04	398	7507.04	398	7558.33	398	7540.05	398	7520.61

Table 1: Results for tuning the number of iterations in the 1-exchange algorithm. The first two columns define the problem instance by the number of jobs and resources, respectively. Then in pairs of columns follow the results for each bound on the number of iterations: 5, 10, 20, 50 and 100. The pair of columns for each bound contains the result value and time usage in seconds.

The results show no obvious pattern. The objective function is not improved regardless the number of iterations. Thus, the algorithm terminates after a couple of iterations and the running times are very similar for all parameter settings. If the objective function should improve then we wish to allow some iterations. However, too many iterations will affect the running time negatively. We set the parameter to 20 iterations.

Next, the heuristic used by the 1-exchange algorithm is determined. The choice stands between the first come first serve, best first, first fit, best fit and random fit heuristics. Results are seen in Table 2.

Generally, the best fit setting finds some of the best result values, but its running time is also among the largest. Using best first and random fit yield low running times, but also low

		FCFS		BFS		FF		BF		RF	
Jobs	Res.	Result	Time	Result	Time	Result	Time	Result	Time	Result	Time
200	10	142	0.20	136	0.16	134	0.23	126	0.43	132	0.08
200	20	154	0.44	154	0.42	152	0.52	152	1.81	150	0.40
200	50	160	3.06	160	2.66	162	3.39	162	11.72	160	1.78
200	100	168	6.70	166	7.56	166	7.83	168	26.54	166	6.92
200	200	194	57.66	194	60.17	194	60.02	194	220.61	192	58.49
200	500	144	215.69	144	233.53	144	221.13	144	809.20	144	221.54
200	1000	162	1272.01	162	1364.04	162	1292.24	162	4833.79	162	1270.27
500	10	290	1.23	260	0.17	288	1.16	270	4.28	260	0.67
500	20	378	3.17	366	2.96	382	3.70	380	11.52	370	2.54
500	50	422	19.87	416	20.16	424	22.99	428	82.96	424	23.41
500	100	426	77.14	430	74.43	426	79.31	430	288.71	424	74.11
500	200	374	276.24	374	130.02	374	253.24	374	1075.54	374	275.42
500	500	426	1501.95	422	663.34	424	1639.85	426	5758.90	426	1514.92
500	1000	398	8936.82	396	7853.65	394	7642.20	396	28291.10	394	7429.52

Table 2: Results for testing the 1-exchange algorithm with the different greedy heuristics as base. The first two columns define the problem instance by the number of jobs and resources, respectively. Then in pairs of columns follow the results for each heuristic: **FCFS** is first come first serve, **BFS** is best first, **FF** is first fit, **BF** is best fit and **RF** is random fit. The pair of columns for each heuristic contains the result value and time usage in seconds.

		5 iter.		10 iter.		20 iter.		50 iter.		100 iter.	
Jobs	Res.	Result	Time	Result	Time	Result	Time	Result	Time	Result	Time
100	10	128	0.04	128	0.05	128	0.04	128	0.05	128	0.04
100	20	78	0.17	78	0.15	78	0.14	78	0.14	78	0.14
100	50	74	0.60	74	0.61	74	0.60	74	0.60	74	0.61
100	100	86	2.35	86	2.35	86	2.34	86	2.37	86	2.40
100	200	70	12.20	70	12.25	70	12.22	70	12.21	70	12.20
100	500	98	56.95	98	57.17	98	57.02	98	56.94	98	57.20
100	1000	72	210.79	72	211.16	72	210.68	72	210.79	72	210.01
200	10	142	0.14	142	0.14	142	0.14	142	0.14	142	0.13
200	20	154	0.34	154	0.33	154	0.33	154	0.33	154	0.34
200	50	160	24.14	160	23.98	160	28.40	160	24.01	160	24.19
200	100	168	6.42	168	6.42	168	6.42	168	6.44	168	6.45
200	200	194	55.46	194	55.98	194	55.95	194	55.46	194	56.01
200	500	144	214.80	144	214.45	144	214.46	144	214.66	144	213.70
200	1000	162	1257.09	162	1261.93	162	1290.93	162	1276.64	162	1262.04

Table 3: Results for tuning the number of iterations in the k-exchange algorithm. The problem instance is given in the first two columns by the number of jobs and resources, respectively. Then in pairs of columns follow the results for each bound on the number of iterations: 5, 10, 20, 50 and 100. The pair of columns for each bound contains the result value and time usage in seconds.

result values. First fit and first come, first serve have similar running times, but first fit gives slightly poorer solution values. We thus choose first come, first serve.

k-exchange algorithm

Similarly to the parameter tuning process for the 1-exchange algorithm, the k-exchange algorithm is tested using the first come first serve heuristic. First, different bounds on the number of iterations are set. The bounds are 5, 10, 20, 50 and 100. Results are seen in Table 3.

The results show that increases in the objective function are not reached and the running time is almost unaltered regardless of the bound on the number of iterations. We choose to set the upper bound on the number of iterations equal to that of the 1-exchange algorithm, 20. We hope that by allowing some iterations the algorithm might be able to improve the objective for some instances.

Jobs	Res.	5 exchanges		10 exchanges		20 exchanges		50 exchanges		100 exchanges	
		Result	Time	Result	Time	Result	Time	Result	Time	Result	Time
100	10	128	0.04	128	0.04	128	0.04	128	0.04	128	0.04
100	20	78	0.14	78	0.14	78	0.14	78	0.14	78	0.15
100	50	74	0.59	74	0.59	74	0.60	74	0.60	74	0.60
100	100	86	2.36	86	2.36	86	2.35	86	2.35	86	2.35
100	200	70	12.36	70	12.20	70	12.31	70	12.26	70	12.26
100	500	98	56.93	98	57.12	98	57.39	98	56.87	98	56.99
100	1000	72	213.30	72	209.81	72	210.91	72	209.73	72	211.21
200	10	142	0.14	142	0.14	142	0.14	142	0.14	142	0.14
200	20	154	0.33	154	0.33	154	0.33	154	0.33	154	0.33
200	50	160	11.36	160	24.06	160	50.48	160	75.64	160	76.14
200	100	168	6.40	168	6.40	168	6.39	168	6.40	168	6.39
200	200	194	55.94	194	55.51	194	55.38	194	55.39	194	55.39
200	500	144	213.97	144	215.47	144	213.99	144	214.44	144	215.41
200	1000	162	1262.53	162	1260.71	162	1515.47	162	1259.71	162	1261.75

Table 4: Results for tuning the maximal number of jobs to exchange at a time in the k-exchange algorithm. The problem instance is given in the first two columns by the number of jobs and resources, respectively. Then in pairs of columns follow the results for each bound on the number of exchanges: 5, 10, 20, 50 and 100. The pair of columns for each bound contains the result value and time usage in seconds.

Jobs	Res.	FCFS		BFS		FF		BF		RF	
		Result	Time	Result	Time	Result	Time	Result	Time	Result	Time
100	10	128	0.04	128	0.03	124	0.30	126	1.10	128	0.17
100	20	78	0.14	78	0.15	76	0.19	78	0.68	78	0.14
100	50	74	0.60	74	0.64	72	0.68	74	2.49	74	0.60
100	100	86	2.37	86	2.56	84	2.60	86	9.46	86	2.36
100	200	70	12.26	70	12.80	70	12.68	70	46.27	70	12.28
100	500	98	56.99	98	62.98	98	65.10	98	229.74	98	57.27
100	1000	72	212.48	72	226.94	72	213.39	72	771.08	72	212.26
200	10	142	0.14	136	0.14	134	0.16	125	2.63	132	0.43
200	20	154	0.33	154	0.34	152	0.40	152	1.44	148	1.64
200	50	160	11.34	160	3.04	162	3.00	162	11.09	156	2.75
200	100	168	6.52	166	6.97	166	7.03	168	25.27	164	33.12
200	200	194	55.53	194	57.59	194	57.84	194	211.88	193	136.73
200	500	144	216.54	144	223.35	144	217.83	144	838.23	144	216.31
200	1000	162	1261.36	162	1300.61	162	1277.72	162	4776.76	162	1263.63

Table 5: Results for testing the k-exchange algorithm with the different greedy heuristics as base. The first two columns define the problem instance by the number of jobs and resources, respectively. Then in pairs of columns follow the results for each heuristic: **FCFS** is first come first serve, **BFS** is best first, **FF** is first fit, **BF** is best fit and **RF** is random fit. The pair of columns for each heuristic contains the result value and time usage in seconds.

The maximal number of jobs to exchange out at a time is investigated. Again, first come first serve is used as heuristic and the bound on the number of iterations is set to 20, according to the parameter tuning above. The settings for the maximal number to exchange at a time are 5, 10, 20, 50 and 100. Results are seen in Table 4.

The results do not show any difference in result values, hence the parameter setting is decided with respect to running times. Here, the 5-exchange setting outperforms the remaining setting and we choose this parameter setting.

Next, the heuristic used by the k-exchange algorithm is determined. Results are seen in Table 5.

The results show that first come, first serve finds the best result values for all instances but one. Furthermore, the first come, first serve has good running times, so we choose this

Jobs	Res.	5 runs		10 runs		20 runs		50 runs		100 runs	
		Result	Time	Result	Time	Result	Time	Result	Time	Result	Time
200	10	136	0.11	136	0.24	136	0.62	136	0.73	136	0.48
200	20	154	0.06	154	0.11	154	0.19	154	0.44	154	1.01
200	50	160	2.41	162	1.76	162	6.29	160	12.36	162	30.58
200	100	166	1.82	166	2.21	166	3.47	166	11.64	166	30.84
200	200	194	6.57	194	15.17	194	28.40	194	87.38	194	249.11
200	500	144	41.21	144	84.96	144	153.03	144	343.98	144	692.90
200	1000	162	242.13	162	403.25	162	733.22	162	1935.13	162	4853.15
500	10	258	0.09	258	0.10	258	0.28	258	0.39	258	2.92
500	20	366	0.30	366	0.27	366	0.48	366	1.27	366	2.72
500	50	426	1.19	416	2.29	418	6.14	416	9.20	416	21.31
500	100	430	3.75	430	7.01	430	11.81	430	24.20	430	91.78
500	200	372	19.35	372	38.45	372	77.76	372	149.88	372	281.87
500	500	420	123.87	420	149.96	426	921.71	422	1418.48	420	1574.94
500	1000	440	439.12	396	755.43	396	1429.54	396	3631.34	396	6605.05

Table 6: Results for tuning the number of heuristic runs per iteration in ALNS. The first two columns define the problem instance by the number of jobs and resources, respectively. Next, pairs of results for each parameter setting are shown. A pair consists of solution value and of time usage in seconds.

heuristic.

6.3.2 The adaptive large neighbourhood search algorithm

The following parameters are to be set:

- *hrun*, the number of times heuristics are run by the algorithm in an iteration.
- k, which is the maximal number of executed jobs to be removed by the remove heuristics.
- ψ_1, the score of finding a new global best solution, ψ_2, the score of finding a new solution with better solution value than the score of the current solution and, ψ_3, the score of finding a new solution with worst solution value than the score of the current solution
- w, the start temperature control parameter
- c, the cooling rate
- r, the reaction factor

To reduce the scope of the parameter tuning process, each parameter is tested individually. The remaining parameters are set to suitable values found through preliminary testing and later by already performed parameter tuning. Initial settings found through preliminary testing are

$$hrun = 50, k = 10, \psi_1 = 60, \psi_2 = 30, \psi_3 = 15, w = 50, c = 1/2 \text{ and } r = 50 \tag{8}$$

Number of heuristic runs per iteration
In each iteration the insertion and removal heuristics are run a number of times to reach an increase in the solution value. The number of heuristic runs is set to 5, 10, 20, 50 and 100, while all other parameters are set according to (8).

The results show that time usage tends to grow proportionally with the upper bound on the number of test runs. This especially holds when the bound goes from 10 or less runs to 20 or more runs. Good solution values are reached even at a bound of 5 runs. The difference

Jobs	Res.	$\|J\|$ runs		$\|J\|/2$ runs		$\|J\|/3$ runs		$\|J\|/4$ runs		$\|J\|/5$ runs	
		Result	Time	Result	Time	Result	Time	Result	Time	Result	Time
200	10	136	0.57	136	0.47	126	0.15	136	0.06	136	0.06
200	20	154	0.27	154	0.26	154	0.24	154	0.20	154	0.21
200	50	160	6.61	162	4.57	160	3.07	162	6.34	160	1.70
200	100	166	10.62	166	9.29	166	18.17	168	6.27	166	3.33
200	200	194	46.57	194	45.28	194	21.04	194	32.32	194	14.62
200	500	144	138.95	144	198.38	144	166.20	144	121.14	144	109.19
200	1000	162	496.95	162	787.10	162	519.09	162	485.74	162	372.52
500	10	258	1.39	258	1.28	258	0.16	258	0.16	258	0.15
500	20	366	0.54	366	0.52	366	1.33	366	2.70	366	1.68
500	50	416	10.55	416	12.31	416	11.14	416	17.93	416	2.80
500	100	430	61.25	430	110.50	430	29.95	430	71.21	430	23.34
500	200	372	106.19	372	173.84	374	139.06	372	118.25	374	76.26
500	500	420	556.01	420	842.79	422	1947.58	422	725.82	420	267.88
500	1000	396	2720.64	396	2420.63	396	2184.90	396	2037.63	396	1073.03

Table 7: The results for tuning the maximal number of jobs to remove at a time in the adaptive large neighbourhood search. The first two columns define the problem instance by the number of jobs and resources, respectively. Next, pairs of results for each parameter setting are shown. A pair consists of solution value and of time usage in seconds.

in solution values is overall small. For these reasons we choose to set the maximal number of runs to 10.

Maximal number of jobs to remove at a time

The removal heuristics takes out a number of executed jobs from the current solution. An upper bound on the number of removed jobs is set via parameter tuning; the tested settings depend on the number of jobs in the instance and are $|J|$, $|J|/2$, $|J|/3$, $|J|/4$ and $|J|/5$. From the previous parameter tuning, we have $hrun = 10$, and the remaining parameters are set according to (8). Results are seen in Table 7.

The results show no obvious pattern. The setting which seems to give shortest running time and good solutions is $|J|/5$. Hence, we choose this parameter setting although other settings would probably be equally good.

Score settings

The score paid to heuristics according to their performance is set via parameter tuning. Four settings are tested:

- **Score 0**: $\psi_1 = 60, \psi_2 = 30$ and $\psi_3 = 15$
- **Score 1**: $\psi_1 = 3, \psi_2 = 2$ and $\psi_3 = 1$
- **Score 2**: $\psi_1 = 100, \psi_2 = 10$ and $\psi_3 = 1$
- **Score 3**: $\psi_1 = 10000, \psi_2 = 100$ and $\psi_3 = 1$

From the previous parameter tuning, we have $hrun = 10$ and $k = |J|/2$. The remaining parameters are set according to (8). Results are seen in Table 8.

Again, a strong pattern does not emerge from the results. There is, however, a small tendency towards best results using setting **Score 0**. Running times for this setting are low and results are good compared to the other settings. Hence, we choose setting **Score 0**: $\psi_1 = 60, \psi_2 = 30$ and $\psi_3 = 15$.

		Score 0		Score 1		Score 2		Score 3	
Jobs	Res.	Result	Time	Result	Time	Result	Time	Result	Time
200	10	136	0.06	136	0.07	136	0.08	136	0.08
200	20	154	0.21	154	0.18	154	0.18	154	0.18
200	50	160	1.70	160	2.85	160	3.20	160	4.80
200	100	166	3.33	166	2.42	166	5.16	166	4.22
200	200	194	14.62	194	14.60	194	28.31	194	22.45
200	500	144	109.19	144	103.83	144	100.03	144	107.87
200	1000	162	273.52	162	405.12	162	456.65	162	447.38
500	10	258	0.15	258	0.15	258	0.16	258	0.16
500	20	366	1.68	366	0.39	366	1.41	366	0.52
500	50	416	2.80	416	6.52	416	6.28	416	3.44
500	100	430	23.34	430	35.74	430	32.79	430	25.25
500	200	374	76.26	372	77.96	372	69.28	374	420.93
500	500	420	267.88	420	345.92	420	293.65	420	523.67
500	1000	396	1073.03	396	1113.50	396	899.38	396	970.74

Table 8: The results for tuning the score to pay for accepted solutions in the adaptive large neighbourhood search. The first two columns define the problem instance by the number of jobs and resources, respectively. Next, pairs of results for each parameter setting are shown. A pair consists of solution value and of time usage in seconds.

		10%		20%		50%		60%		100%	
Jobs	Res.	Result	Time	Result	Time	Result	Time	Result	Time	Result	Time
200	10	136	0.22	136	0.20	136	0.20	136	1.05	136	0.06
200	20	154	0.17	154	0.18	154	0.18	154	0.16	154	0.18
200	50	160	4.11	158	1.11	162	2.33	160	2.90	160	1.89
200	100	166	3.88	168	6.10	166	4.82	166	3.44	166	3.09
200	200	194	29.24	194	22.94	194	21.90	194	33.34	194	21.56
200	500	144	118.42	144	102.54	144	126.34	144	114.20	144	87.80
200	1000	162	499.94	162	549.69	162	598.98	162	519.71	162	473.00
500	10	258	0.15	258	0.15	258	0.15	258	0.15	258	0.15
500	20	366	2.61	366	0.51	366	0.58	366	0.54	366	0.45
500	50	416	7.12	416	8.20	416	5.44	416	4.22	416	8.28
500	100	430	42.89	430	17.41	430	22.76	430	23.45	430	32.29
500	200	372	111.65	374	266.37	372	68.28	372	129.05	374	128.45
500	500	420	383.02	420	561.29	420	303.62	420	210.70	420	433.85
500	1000	396	1122.42	396	1111.02	396	1435.85	396	1667.41	396	2042.09

Table 9: The results of tuning the start temperature control parameter in the adaptive large neighbourhood search algorithm. The first two columns define the problem instance by the number of jobs and resources, respectively. Next, pairs of results for each parameter setting are shown. A pair consists of solution value and of time usage in seconds.

Start temperature control parameter

The start temperature control parameter, w, is to be set. This parameter says that a solution, which is w percent worse than the last solution is accepted with a probability of 50%. Settings for w are 10%, 20%, 50%, 60% and 100%. From the previous parameter tuning, we have $hrun = 10, k = |J|/2$ and $\psi_1 = 60, \psi_2 = 30$ and $\psi_3 = 15$. The remaining parameters are set according to (8). Results are seen in Table 9.

Once again, we do not see a strong pattern, but the results values are of good quality and running times are low for tests with setting $w = 50\%$. For these reasons, we choose this setting.

Cooling rate

The cooling rate c, is set. The cooling rate is a part of the simulated annealing process of the ALNS. Settings for c are 0.1, 0.17, 0.2, 0.5 and 1. From the previous parameter tuning, we have $hrun = 10, k = |J|/2, \psi_1 = 60, \psi_2 = 30, \psi_3 = 15$ and $w = 50\%$. The remaining parameters are set according to (8). Results are seen in Table 10.

		0.1		0.17		0.2		0.5		1	
Jobs	Res.	Result	Time	Result	Time	Result	Time	Result	Time	Result	Time
200	10	136	0.11	136	0.10	136	0.10	136	0.10	136	1.82
200	20	154	0.27	154	0.27	154	0.28	154	0.30	154	0.29
200	50	160	10.00	160	5.25	160	5.22	160	6.96	160	12.92
200	100	166	11.43	166	6.26	166	15.78	166	13.08	166	9.37
200	200	194	58.23	194	49.37	194	60.80	194	40.58	194	67.98
200	500	144	165.57	144	178.50	144	152.77	144	205.78	144	219.17
200	1000	162	1127.33	162	873.56	162	1115.58	162	779.73	162	879.05
500	10	258	0.36	258	0.28	258	0.28	258	2.29	258	3.81
500	20	366	5.70	366	3.76	366	0.95	366	1.02	366	2.47
500	50	416	10.04	416	23.02	416	9.47	416	11.08	416	14.05
500	100	430	34.63	430	55.33	430	38.07	430	87.61	430	52.96
500	200	372	182.27	372	227.90	372	182.14	372	209.91	374	365.74
500	500	420	728.78	420	637.98	420	520.13	420	719.08	420	576.30
500	1000	396	2370.68	396	3471.70	396	2464.27	396	2901.12	396	2238.43

Table 10: Results of the parameter tuning of the cooling rate,c. The first two columns of the table define the problem instance by the number of jobs and resources, respectively. Next, pairs of results for each parameter setting are shown. A pair consists of solution value and of time usage in seconds.

		0.1		0.17		0.2		0.5		1	
Jobs	Res.	Result	Time	Result	Time	Result	Time	Result	Time	Result	Time
200	10	136	0.05	136	0.32	136	0.80	136	1.12	136	0.58
200	20	154	0.14	154	0.15	154	0.18	154	0.15	154	0.14
200	50	160	1.67	160	4.66	158	1.16	162	1.69	160	12.22
200	100	168	6.21	166	3.24	166	5.86	166	8.47	166	5.82
200	200	194	20.18	194	34.35	194	33.76	194	28.45	194	21.12
200	500	144	139.52	144	101.06	144	131.95	144	141.30	144	128.30
200	1000	162	456.45	162	596.19	162	453.44	162	545.42	162	607.01
500	10	258	1.27	258	3.19	258	0.20	258	0.20	258	0.22
500	20	366	2.91	366	1.94	366	0.44	366	0.45	366	0.53
500	50	416	6.45	416	5.17	416	4.26	416	2.76	416	3.61
500	100	430	28.75	430	18.03	430	27.54	430	27.24	430	16.68
500	200	374	105.75	372	81.61	374	102.91	374	350.32	372	111.48
500	500	420	317.96	424	641.06	420	361.78	420	346.82	426	476.61
500	1000	396	1239.50	396	987.91	396	1225.37	396	1270.17	396	1151.23

Table 11: Results for setting the reaction factor, r, to different values. The first two columns define the problem instance by the number of jobs and resources, respectively. Next, pairs of results for each parameter setting are shown. A pair consists of solution value and of time usage in seconds.

The results again do not show any clear pattern. The setting 1, however, gives both good result values and has good running times. We thus choose to set the cooling rate to 1.

Reaction factor

The reaction factor r, must also be chosen. Settings for r are 0.1, 0.17, 0.2, 0.5 and 1. From the previous parameter tuning, we have $hrun = 10, k = |J|/2, \psi_1 = 60, \psi_2 = 30, \psi_3 = 15, w = 50\%$ and $c = 1$. Results are seen in Table 11.

Best result values are reached by using reaction factor 1. The running times for this setting are generally higher than for the other settings. Setting 0.17 reaches good results and the running times are generally low. We thus choose this setting.

6.4 Results

Using the parameter settings found in the previous section, all heuristics are finally tested and compared. Note, that in this final computational evaluation, a two hour upper bound is

Heuristic	Worst %	Aver. %
First come, first serve	28.57	20.92
Best first	28.57	20.40
First fit	28.57	21.60
Best fit	29.55	20.15
Random	28.57	20.35
1-exchange	28.57	18.00
k-exchange	28.57	18.00
ALNS	28.57	19.24

Table 12: Gaps between heuristic and optimal values are calculated. In the first column the name of the heuristic to compare is given. The second column contains the largest gap between the solution of the heuristic and the optimal solution. The gap is given in percent. The third column contains the average gap between the solution of the heuristic and the optimal solution. Again, the gap is given in percent.

set for the running times. Results are compared with respect to solution values and to time usage.

Detailed results for the computational evaluation of the greedy heuristics are seen in Table 13 and detailed results for the local search heuristics and the ALNS are seen in Table 14.

First, solution values for some smaller instances are compared to the optimal solution values. The optimal solution values are found by a branch-and-price algorithm presented in [10] and can be seen in Table 15. The average and the worst gap between the optimal and the found solution values are gathered in Table 12. The three more sophisticated algorithms generally find better solutions than the greedy heuristics. The largest gap between a heuristic and an optimal solution, however, is the same for almost all heuristics. The largest gap is never more than 30%, and the average gap is from 18% to 22%. Hence, all heuristics perform well with respect to solution values.

Next, running times are compared. Time usage is sorted according to the number of jobs, e.g. running times for all first fit test runs on instances with 100 jobs are added and then divided with 7 (there are 7 instances with 100 jobs). The running times sorted according to the number of jobs are seen in Figure 2.

Generally, time usage grows with the number of jobs. This is expected due to the corresponding theoretical running times. Comparing all heuristics with each other, the first come first serve and the random fit heuristics have the smallest practical running times followed by the first fit and the best first heuristics. The best fit heuristic has the largest practical time usage of the greedy heuristics. The three sophisticated heuristics have much larger practical time usage than the greedy heuristics. Furthermore, the local search algorithms have larger practical running times than the ALNS.

Theoretical running times are presented using $\mathcal{O}$-notation at the introduction of each heuristic in Sections 3 and 4. The practical time usage reflects the theoretical running times well. The first come first serve, the random fit and the first fit heuristics have the same theoretical running time, which is also the smallest. Test results show that the first fit heuristic obviously has some overhead not included by the theoretical time usage. Next, the best first heuristic has smaller theoretical time usage than the best fit, which corresponds to the test results. The sophisticated heuristics all have larger theoretical running times than the greedy

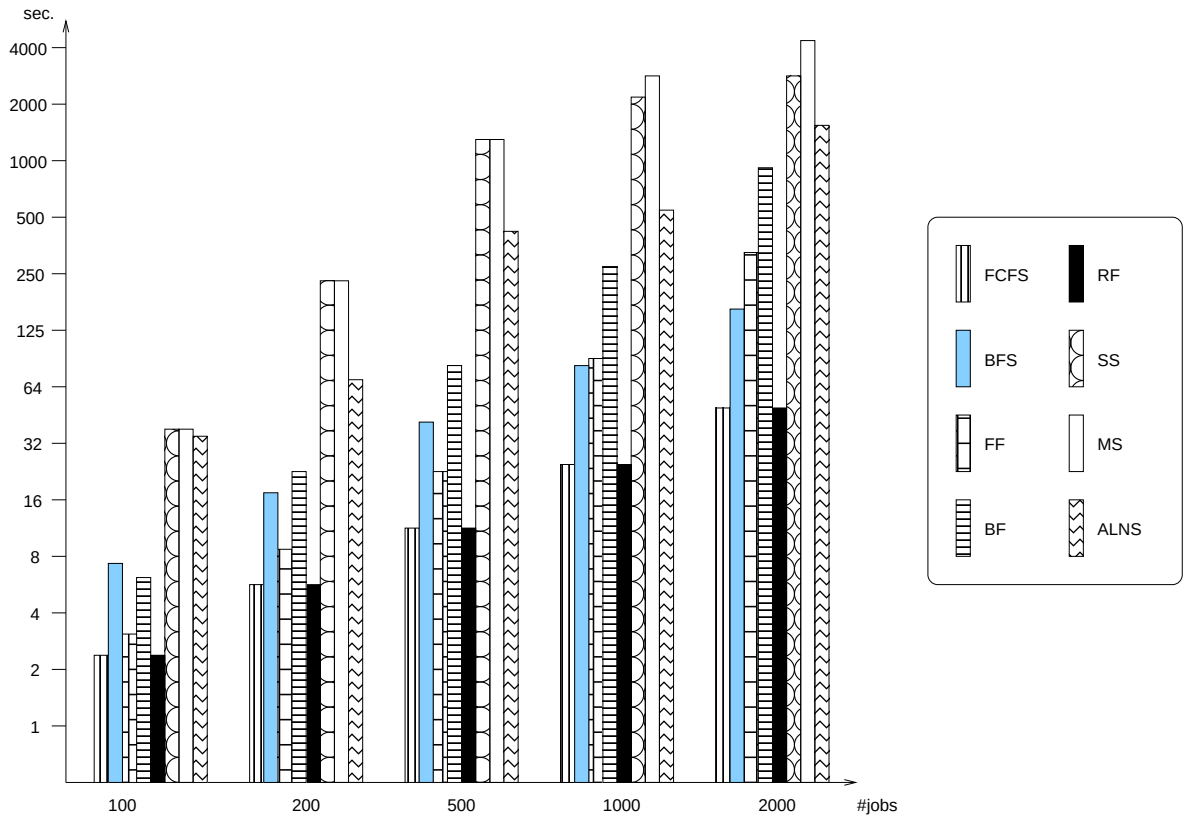


Figure 2: Time usage according to the number of jobs. For each setting of the number of jobs and for each heuristic, the average time usage of the instances is calculated. The x-axis runs over the number of jobs. The y-axis denotes time usage in seconds; note that the axis is logarithmic. The box on the right-hand-side show which column corresponds to which heuristic. FCFS is short for the first come first serve heuristic, BFS for the best first, FF for the first fit, BF for the best fit, and RF for the random fit heuristic. Furthermore, SS is short for the 1-exchange heuristic, MS for the k-exchange heuristic, and ALNS for adaptive large neighbourhood search.

heuristics. The theoretical running times of the sophisticated heuristics are somewhat similar. The practical running times for the local search heuristics are, however, larger than those of the ALNS. For several instances, the k-exchange heuristic is even stopped by the upper bound on time usage. The reason for the larger running times of the local search heuristics must be due to overheads not included by the theoretical running times.

Finally, both time usage and solution values are considered. If time usage is the most prominent factor for choosing a heuristic, it is recommended to choose one of the greedy heuristics because they are capable of finding good solutions in relatively little time. If the quality of the solution is the primary priority then the 1-exchange and the k-exchange local search heuristics appear to perform best. We recommend the 1-exchange local search heuristic, because it has smaller time usage. Generally, the trade off lies between time usage and solution quality. If both are equally important then a reasonable compromise must be found.

Looking at the suggested usage of the offline heuristics, we expect that time usage may not be that important when analyzing the results of changing the components of the grid. When investigating whether or not some planned jobs can be executed, a (greedy) heuristic can be chosen to give a fast, but possibly inaccurate answer, and another (more sophisticated) heuristic can be chosen to give a more accurate result later on. Finally, when emptying the queue, the grid administrators most likely prefer having a plan straight away and may thus choose a fast heuristic. Hence, the implemented heuristics all have benefits and are usable despite their possible drawbacks.

7 Final remarks

In this article, we have analyzed the offline scheduling problem in grid computing. The problem is formally described and a complexity proof of the $\mathcal{NP}$-hardness of the problem is given.

To solve the offline scheduling problem a number of heuristics are implemented and tested. Five of the heuristics are of greedy nature, two are local search algorithms and the final is an adaptive large neighbourhood search algorithm. The two local search algorithms can use any of the greedy heuristics as base, and thus the actual number of implemented heuristics is 16. Through parameter tuning, however, we limit the local search algorithms to only use one greedy heuristic each. The adaptive large neighbourhood search algorithm is parameter tuned to proper values.

A computational evaluation is performed. The test results show that the three more sophisticated heuristics find very good solutions at the cost of a large time consumption. The five greedy heuristics generally find solutions of good quality and has significantly smaller time consumption.

The heuristics thus have different advances and drawbacks. Overall they all perform well, and are very usable in the suggested areas of offline scheduling usage. Whenever a fast result is needed, e.g., to empty the job queue or to generate a fast overview of the execution of planned jobs, the greedy heuristics will give a good solution in little time. When performing in-depth analysis of grid performance, e.g., when investigating the results of adding or removing resources, then the local search algorithms and the adaptive large neighbourhood search should be used to reach high quality solutions.

The final conclusion is thus that the heuristics can easily be used both in practice for reserving resources and emptying the job queue and as analytic tools to investigate and measure grid performance.

		FCFS		BFS		FF		BF		RF	
Jobs	Res.	Result	Time	Result	Time	Result	Time	Result	Time	Result	Time
100	10	124	0.00	128	0.00	116	0.05	126	0.20	128	0.00
100	20	78	0.00	78	0.02	74	0.06	78	0.19	78	0.01
100	50	74	0.03	74	0.12	72	0.12	74	0.38	74	0.03
100	100	86	0.11	86	0.43	84	0.37	86	1.00	86	0.11
100	200	70	0.62	70	1.63	70	1.01	70	2.78	70	0.63
100	500	98	2.74	98	12.02	98	4.49	98	10.46	98	2.87
100	1000	72	14.00	72	40.78	72	15.97	72	33.41	72	13.90
200	10	126	0.00	136	0.01	126	0.08	124	0.21	132	0.00
200	20	154	0.02	154	0.03	146	0.17	152	0.49	150	0.02
200	50	158	0.10	158	0.24	162	0.56	162	1.56	160	0.10
200	100	168	0.27	166	0.88	164	1.13	168	3.14	168	0.27
200	200	190	1.38	194	4.13	194	3.76	194	10.81	194	1.32
200	500	144	6.74	144	20.26	144	11.08	144	26.78	144	6.61
200	1000	162	33.77	162	93.19	162	43.76	162	111.96	162	33.88
500	10	256	0.02	258	0.02	260	0.35	268	1.18	260	0.02
500	20	366	0.05	366	0.09	358	1.02	378	3.25	368	0.05
500	50	426	0.26	416	0.62	410	3.37	428	11.04	422	0.26
500	100	424	0.84	430	2.53	420	7.06	430	22.64	422	0.85
500	200	374	3.18	372	8.68	372	13.55	374	41.98	374	3.20
500	500	426	16.52	420	56.74	422	46.13	426	136.97	424	16.76
500	1000	394	67.68	396	219.27	394	122.26	396	363.03	392	68.83
1000	10	318	0.04	320	0.06	370	1.24	328	3.65	324	0.04
1000	20	736	0.17	754	0.26	804	6.21	768	20.53	770	0.17
1000	50	788	0.60	794	1.20	772	12.14	816	40.81	802	0.58
1000	100	830	1.90	856	4.86	824	26.36	860	92.66	834	1.93
1000	200	828	6.09	830	18.25	820	51.60	836	173.81	824	6.10
1000	500	806	35.80	806	109.74	804	143.53	812	467.79	808	35.90
1000	1000	814	131.95	814	437.47	812	348.52	814	1038.78	814	134.84
2000	10	436	0.14	444	0.18	506	5.34	458	16.81	432	0.12
2000	20	620	0.30	598	0.41	704	9.47	630	29.19	620	0.30
2000	50	1572	1.78	1580	2.68	1604	55.28	1624	189.17	1558	1.76
2000	100	1686	5.05	1688	10.25	1642	113.05	1750	386.58	1682	5.00
2000	200	1444	12.65	1432	31.93	1396	154.45	1456	528.67	1436	12.51
2000	500	1612	79.75	1616	221.72	1566	526.19	1622	1733.00	1612	78.78
2000	1000	1706	310.21	1706	938.17	1696	1324.38	1719	4010.46	1706	313.94

Table 13: Results for testing the greedy heuristics. Problem instances are given in the first two columns consisting of the number of jobs and resources, respectively. **FCFS** stands for first come first serve, **BFS** for best first, **FF** for first fit, **BF** for best fit and **RF** for random fit. A pair is given for each of the five algorithms. A pair consists of result value and time usage in seconds.

		SS		MS		ALNS	
Jobs	**Res.**	**Result**	**Time**	**Result**	**Time**	**Result**	**Time**
100	10	128	0.05	128	0.05	128	0.12
100	20	78	0.16	78	0.14	78	0.27
100	50	74	0.61	74	0.60	74	0.74
100	100	86	2.51	86	2.34	86	1.87
100	200	70	12.31	70	12.32	70	9.56
100	500	98	57.97	98	56.77	98	50.53
100	1000	72	212.86	72	209.55	72	180.05
200	10	142	0.17	142	0.14	136	0.07
200	20	154	0.41	154	0.33	154	0.22
200	50	160	3.01	160	11.42	158	1.04
200	100	168	6.68	168	6.34	166	3.31
200	200	194	57.09	194	55.29	194	14.03
200	500	144	218.75	144	213.00	144	120.06
200	1000	162	1265.29	162	1254.15	162	342.57
500	10	290	1.24	290	3.36	258	0.16
500	20	378	3.14	378	14.05	366	0.42
500	50	422	20.10	422	92.34	416	3.63
500	100	426	76.76	426	68.53	430	32.74
500	200	374	274.59	374	271.96	374	135.76
500	500	426	1490.83	426	1472.28	422	1102.17
500	1000	396	7228.42	398	7070.13	396	1579.52
1000	10	372	6.22	372	12.72	320	6.34
1000	20	820	36.04	820	95.25	754	9.95
1000	50	802	90.48	802	385.03	794	21.63
1000	100	848	311.49	848	1550.22	856	70.50
1000	200	828	1175.01	828	5929.71	830	276.04
1000	500	810	6116.43	810	5851.33	806	1231.17
1000	1000	812	7242.48	812	7254.15*	814	2487.58
2000	10	506	31.88	506	46.87	444	1.58
2000	20	710	71.26	710	113.72	598	58.68
2000	50	1590	610.32	1590	1954.48	1580	115.08
2000	100	1688	1851.08	1688	7202.08*	1688	384.21
2000	200	1432	4043.54	1432	7200.41*	1432	678.04
2000	500	1624	7208.33	1624	7228.64*	1616	3050.74
2000	1000	1706	7266.57	1706	7245.76*	1706	6894.21

Table 14: Results for testing the local search algorithms and the adaptive large neighbourhood search. Problem instances are given in the first two columns consisting of the number of jobs and resources, respectively. **SS** stands for the 1-exchange algorithm, **MS** for the k-exchange algorithm and **ALNS** for the adaptive large neighbourhood search. A pair is given for each of the three algorithms. A pair consists of result value and time usage in seconds.

Jobs	**Resources**	**Opt. val**
100	10	136
100	20	104
100	50	94
100	100	106
100	200	98
200	10	176
200	20	182
200	50	206
200	100	198
500	10	358
500	20	462
500	50	518
1000	10	398

Table 15: Optimal results for some of the test instances. The selected instances are those, which the branch-and-price algorithm are capable of solving, see [10].

Acknowledgement
We would like to thank GlobalConnect A/S for their support of this work.

References

[1] V. Agarwal, G. Dasgupta, K. Dasgupta, A. Purohit, and B. Viswanathan. Deco: Data replication and execution co-scheduling for utility grids. In *ICSOC 2006, LNCS 4294*, pages 52–65, 2006.

[2] R. K. Ahuja, T. L. Magnanti, and J. B. Orlin. *Network Flows: Theory, Algorithms, and Applications*. Prentice-Hall Inc., 1993.

[3] R. Andersen and B. Vinter. Direct application access to grid storage. *Concurrency and Computation: Practice and Experience*, 19(9):1287–1298, 2007.

[4] V. Arya, N. Garg, R. Khandekar, A. Meyerson, K. Munagala, and V. Pandit. Local search heuristics for k-median and facility location problems. *Society for Industrial and Applied Mathematics (SIAM)*, 33(3):544–562, 2004.

[5] B. Awerbuch and T. Leighton. *Multicommodity flows: A survey of recent research*, volume 762 of *Lecture Notes in Computer Science*, pages 297–302. Springer Berlin / Heidelberg, 1993.

[6] M. Baker, R. Buyya, and D. Laforenza. Grids and grid technologies for wide-area distributed computing. *Software -practice and experience*, 32(15):1437–1466, 2002.

[7] A. Elghirani, R. Subrata, and A. Y. Zomaya. Intelligent scheduling and replication in datagrids: a synergistic approach. In *Seventh IEEE International Symposium on Cluster Computing and the Grid (CCGrid'07)*, 2007.

[8] L. R. Ford and D. R. Fulkerson. Constructing maximal dynamic flows from static flows. *Operations Research*, 6(3):419–433, 1958.

[9] L. R. Ford and D. R. Fulkerson. *Flows in Networks*. Princeton University Press, Princeton, New Jersey, 1962.

[10] M. Gamst and D. Pisinger. Integrated job scheduling and network routing. In submission, September 2009.

[11] J. Kennington. A survey of linear cost multicommodity network flows. *Operations Research*, 26:209–236, 1978.

[12] T. Larsson and D. Yuan. An augmented lagrangian algorithm for large scale multicommodity routing. *Computational Optimization and Applications*, 27(2):187–215, 2004.

[13] L. Marchal, Y. Robert, P. V.-B. Primet, and J. Zeng. Scheduling network requests with transmission window. Technical Report 2005-32, Laboratoire de L'Informatique du Parallélisme, 2005. July.

[14] D. Pisinger. *Algorithms for Knapsack Problems*. PhD thesis, Dept. of Computer Science, University of Copenhagen, Febraury 1995.

[15] S. Røpke and D. Pisinger. An adaptive large neighborhood search heuristic for the pickup and delivery problem with time windows. Technical Report 04-13, DIKU, University of Copenhagen, Denmark, 2004.

[16] R. B. Sørensen. Analysing resource allocation in minimum intrusion grid (mig) using mechanism design. Master's thesis, Department of Computer Science, Copenhagen University (DIKU), 2007.

[17] E. Varvarigos, V. Sourlas, and K. Christodoulopoulos. Routing and scheduling connections in networks that support advance reservations. *Computer Networks*, 52:2988–3006, 2008.

[18] B. Vinter. The architecture of the minimum intrusion grid, mig. In *Communicating Process Architecture*, pages 189–201, 2005.

In grid computing a number of geographically distributed resources connected through a wide area network, are utilized as one computations unit. The NP-hard offline scheduling problem in grid computing consists of assigning jobs to resources in advance. In this paper, five greedy heuristics and two metaheuristics for solving the offline scheduling problem are introduced. Computationally evaluating the heuristics shows that all heuristics find useful solutions with a gap of 20\% between upper and lower bounds. The metaheuristics give better results than the greedy heuristics, but also have larger time usage. All heuristics solve instances with up to 2000 jobs and 1000 resources, thus the results are useful both with respect to running times and to solution values.

ISBN 978-87-90855-65-9

DTU Management Engineering
Department of Management Engineering
Technical University of Denmark

Produktionstorvet
Building 424
DK-2800 Kongens Lyngby
Denmark
Tel. +45 45 25 48 00
Fax +45 45 93 34 35

www.man.dtu.dk